Ch. 2 Math Preliminaries for Lossless Compression

Section 2.2 Info Theory

Section 2.3 Models

Motivation for Info Theory & Models

- Gives Mathematical Foundation for Compression
  - How do we mathematically describe how much "information" is in "data"?
  - How do we model information and data?
- Provides Theoretical Limits for Compression
  - For a given type of data, what is the smallest number of bits that can be used to represent it?
  - What aspects of the data impact this lower bound?
- Motivates Practical Algorithms for Compression
  - Theoretically, what aspects of data can we exploit?
  - What kinds of processing is best?
  - How close do the real-world algorithms come to the theoretical limits

Section 2.2
Information Theory

<u>Defining Information</u>

Needed Characteristics:

- Probability is involved…
  - <u>Rare</u> events… convey <u>large</u> amounts of info
  - <u>Common</u> events… convey <u>small</u> amounts of info
- Info is additive for <u>*independent*</u> events
  - $i(A \text{ and } B) = i(A) + i(B)$

➔ **Define Info of an <u>Event</u> *A*:**

$$\boxed{\begin{aligned} i(A) &= \log_2\left[\frac{1}{P(A)}\right] \quad (bits) \\ &= -\log_2\left[P(A)\right] \end{aligned}}$$

It is easy to verify that this definition satisfies the two "needed characteristics" above

Simplest Form of Avg. Info of a Source (Entropy)

➔ Each source symbol conveys some amount of info – generally, not all symbols convey the same amount of info

Q: On *average*, how much info does a source put out per symbol

A: First consider a source S putting out a stream of symbols (i.e., RVs) that are **I**ndependent & **I**dentically **D**istributed (iid)…

Assume source "alphabet" consists of symbol set $\{A_1, A_2, \ldots, A_N\}$

Recall that the info of the k^{th} symbol is $i(A_k) = -\log_2[P(A_k)]$

Then… the source's ***average*** info is just the average of the symbol informations…

$$\text{Source's Avg. Info} = E\{i(A_i)\} = \sum_{i=1}^{N} i(A_i)P(A_i)$$

"Entropy of Source"

For iid source:
$$\boxed{H(S) = -\sum_{i=1}^{N} P(A_i)\log_2\left[P(A_i)\right]} \quad \text{(bits/symbol)}$$

Entropy of the source = Avg. Info conveyed per symbol (in bits)

More General Form of Entropy of a Source

But… most real sources are **NOT** independent

(Recall: English text… prob. of next letter depends on current letter)

So… need to capture the (possibly infinite-order) dependence between subsequent symbols emitted by a source

We'll use an asymptotic (or limiting) approach…

Define 2nd-Order Entropy… "Avg. Info/Pair of Symbols"

$$\boxed{G_2(S) = -\sum_{i=1}^{N}\sum_{j=1}^{N} P(A_i, A_j)\log_2\left[P(A_i, A_j)\right]}$$ (bits/symbol-pair)

Define 3rd-Order Entropy… "Avg. Info/Triple of Symbols"

$$\boxed{G_3(S) = -\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} P(A_i, A_j, A_k)\log_2\left[P(A_i, A_j, A_k)\right]}$$ (bits/symbol-triple)

•

• **Etc. 4th, 5th, 6th, …**

•

Next

Now… Need to convert each of these into a "per symbol" form:

Define: $$\boxed{H_n(S) = \frac{1}{n} G_n(S)}$$ (bits/symbol)

"n^{th}-order Entropy"

As we increase *n* we capture <u>more and more</u> of the <u>inter-symbol structure</u>

In the limit we capture **<u>all</u>** the structure…

Define the Entropy of the Source *S* to be:

$$\boxed{H(S) = \lim_{n\to\infty} H_n(S) = \lim_{n\to\infty} \frac{1}{n} G_n(S)}$$ (bits/symbol)

(For General Source)

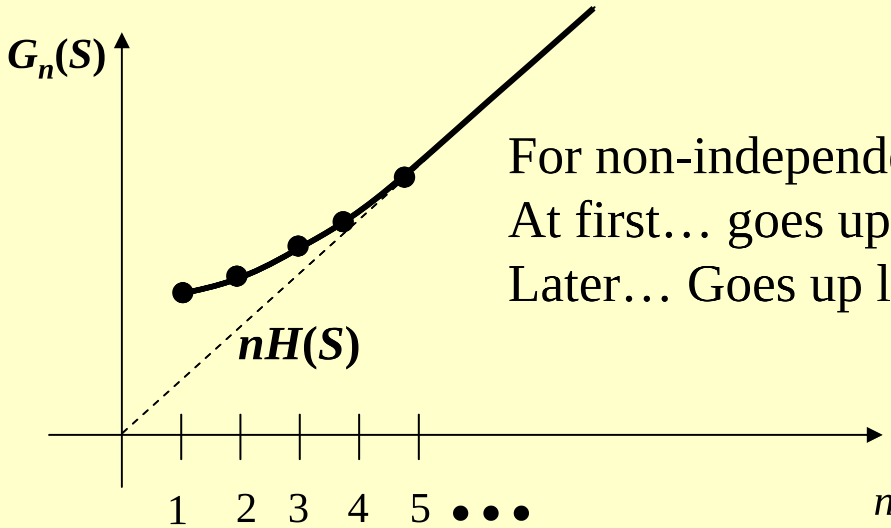
Can verify that ***<u>if</u>*** the ***<u>source is iid</u>***, then this general entropy collapses to

$$\boxed{H(S) = \lim_{n\to\infty} H_n(S) = -\sum_{i=1}^{N} P(A_i)\log_2\left[P(A_i)\right]}$$

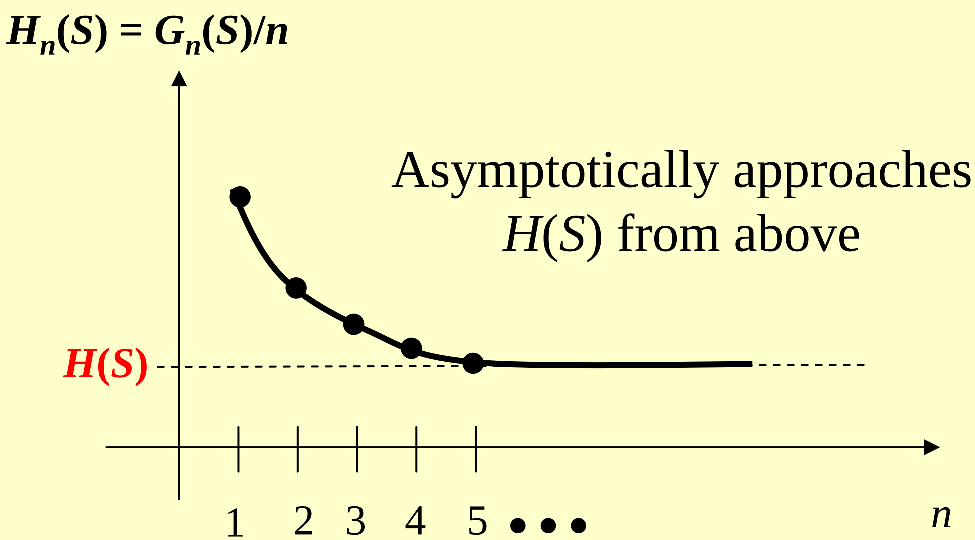
Same as we defined before for iid!

Next

Typical Behavior of General Form of Entropy



For non-independent source:
At first… goes up slower than linear
Later… Goes up linearly with slope of H(s)



A decreasing "return" as the order increases!

Next

<u>Two Views of Entropy $H(S)$</u>

- Expresses the Avg Info per symbol conveyed by source
  - Although each symbol conveys a different amount of information, $H(S)$ gives the overall average amount
  - Sources with the same number of symbols can have different entopy values
- $H(S)$ gives a Lower Bound on the average # of bits/symbol needed to code the source
  - This provides a measure of what the best level of compression we can expect for a given source

<u>Recall</u> "Lossless Example to Motivate…"

If the source is assumed to be iid, with symbol probabilities of

$P(A) = 0.5$ $\quad$ $P(B) = 0.25$ $\quad$ $P(C) = 0.125$ $\quad$ $P(D) = 0.125$

Then $H(S) = 1.75$ bits… which is exactly what our best decodable code achieved!!!

So… $H(S)$ tells us exactly the <u>lowest rate</u> the <u>*best*</u> lossless compression scheme can compress <u>source S</u> to!!!

Next

Section 2.3
Models

Type of Models

What kind of models can capture the prob. structure of a source?

1. **Prob. Model**... Can't capture symbol-to-symbol dependence
   - Symbols: $A_1, A_2, \ldots, A_N$
   - Probabilities: $P(A_1), P(A_2), \ldots, P(A_N)$

2. **Joint Prob. Model**... Can capture sym-to-sym dependence
   - **2nd-Order Model**:
     $P(A_1,A_1), P(A_1,A_2), \ldots, P(A_1,A_N), P(A_2,A_1), \ldots, P(A_2,A_N), \ldots, P(A_N,A_N)$
   - **3rd-Order Model**:
     $P(A_1,A_1,A_1), P(A_1,A_1,A_2), \ldots, P(A_1,A_1,A_N), \ldots, P(A_N,A_N,A_N)$
   - **Etc., Etc., Etc.…**

1st-Order Model? **Just a Prob. Model…**
… Called 1st-Order Prob Model

Next

3. **Cond. Prob. Model**... Can capture sym-to-sym dependence

Called a "Context Model"… Most useful model of the three…

A special Cond. Prob. Model is called a "Markov Model" (MM) which is also called a "Discrete-Time Markov Chain"

Suppose that the source output sequence is $x_1, x_2, x_3, \ldots$
and each x_i can take on any symbol $A_1, A_2, \ldots, A_N$.

– **1st-Order MM – Context beyond one symbol has no effect**:

$$P(x_n | x_{n-1}, \underbrace{x_{n-2}, x_{n-3}, x_{n-4}, \ldots}_{\text{Extra context has no effect}}) = P(x_n | x_{n-1})$$

– ***k*th-Order MM**:

$$P(x_n | x_{n-1}, x_{n-2}, \ldots, x_{n-k}, x_{n-(k+1)}, \ldots) = P(x_n | x_{n-1}, x_{n-2}, \ldots, x_{n-k})$$

Note: A 0th-Order MM is just a 1st-Order Prob. Model

If source really does have only finite context, what is its entropy?

For 1st-Order MM the result of finding $H(S)$ via the limiting approach is:

$$\boxed{H(S) = -\sum_{i=1}^{N}\sum_{j=1}^{N} P(A_i, A_j)\log_2\left[P(A_i | A_j)\right]}$$

Next

Using MMs for Compression

How does one use a k^{th}-Order MM for compression?

For an N symbol alphabet… build a set of N codewords for each k-symbol context… Called "Context Coding"

There will be N^k contexts… so there will be N^{k+1} codewords!

In building compression algorithms… there is a trade-off:

model complexity vs. model accuracy

Higher-order MM more accurately captures source structure…

… BUT… increases the number of codewords

Next

Example: Context Coding via 1^{st}–Order MM

Consider in English text… how should one code the letter *u*?

It depends on the context…

If current letter = *q*… then next letter is very likely *u*…

➔ in this context the codeword for *u* should be short

But… if current letter = *u*… then next letter is unlikely *u*…

➔ in this context the codeword for *u* should be long

Note that the decoder – once it decodes the current symbol – can choose the correct set of codewords to decode the next symbol

Example: 3-symbol alphabet, source model is 1^{st}-Order MM

(Assume that repeated symbols are not likely to occur)

Current Symbol / Next Symbol	A_1	A_2	A_3
A_1	1	00	01
A_2	00	1	01
A_3	00	01	1

Codewords for next symbol conditioned on current symbol

Next

Demo of Text Generation Via Models

A model that accurately captures a source's conditional probability structure is useful for compression… a way to visualize how well a model captures this structure is to use the model to generate a symbol sequence and see if it "looks right"

This is most easily visually checked for the case of English text…

- For 1st-Order Prob. Model
  - Use computer to analyze large sample of text to estimate prob. of each symbol: P(a) = (# of a's)/(total # of characters)
  - Then… generate stream of symbols drawn according to measured prob.
- For k^{th}-Order MM
  - Use computer to analyze large sample of text to estimate prob. of each symbol under each k-symbol context: for example, for $k = 1$

    P(a|b) = (# of a's preceded by b)/(total # of characters preceded by b)

Next

“0th-Order” Probability Model
(i.e., Assume equal probabilities for a-z & space)

```
>> char(gen_text_0(70))

tgpgmmvi lhaedxyx wzgkpdsufeflmcqsjehhantyxmmwzwxuxghmimiixdbmrvsiionp
```

```
>> char(gen_text_0(70))

xpcnv gvaelshgvbprabbdzcvrve qmganfcqddrzdiycxdbkhyrkjdzklq vbtd jhrcw
```

```
>> char(gen_text_0(70))

dsxuwnqjvnywokiaxdttjmtjwcmotln oqcxbkzllwshhifwkvjs lsoazhazmlrzshjfd
```

```
>> char(gen_text_0(70))

andjmaxgxkphlqitpjwugpt ahfqfklahakcwmzpyrgfwglnrnsicojmhthixmtwjxtmrt
```

```
>> char(gen_text_0(70))

dwndroffhdpbjfxukvvpqwodrsaqwiquylyxvntkoupisqlcamrjxycstrpkzqupkpnxyo
```

1^{st}-Order Probability Model
(i.e., estimated probabilities for a-z, A-Z & space)

```
>> char(gen_text(probs,70))

oe pciA otwcrfa art bb fw iufnktl uibldt is slPe  d sto faeb  Lldeo st
```

```
>> char(gen_text(probs,70))

lka  lo ekosa eocekdptbcfy manoAaomrxe riu c cs eieucl Za cs impiakoj
```

```
>> char(gen_text(probs,70))

pjd dot stlAa o ue gil    oe k  sl ourls islllaelsadu m g  snb   yh e
```

```
>> char(gen_text(probs,70))

pjd dot stlAa o ue gil    oe k  sl ourls islllaelsadu m g  snb   yh e
```

```
>> char(gen_text(probs,70))

x Hcnd  ceh eelacoulsbrke atmitr kun i ekes tal eaddp   nootltiNot iac
```

1st-Order Markov Model – Conditioned on Single Letter
(i.e., estimated cond. probs. for a-z, A-Z & space)

```
>> char(gen_text_2(cond_probs_2,70))

ho lenaf b me t chontha ffes ntarub fulep eot Errol sibl uxiritha fte
```

```
>> char(gen_text_2(cond_probs_2,70))

flug cee Roly nararme sp grirme sork d ak focamase asnolot bartan hov
```

```
>> char(gen_text_2(cond_probs_2,70))

se fu slovelop t ta d gi Morn puea schel be friilgud a d pe Prilabe b
```

```
>> char(gen_text_2(cond_probs_2,70))

cmel wntoi aga f bube mewil ceng g Dave gilar peedeazy st h wl cist m
```

```
>> char(gen_text_2(cond_probs_2,70))

lly Munebest hn rte leara scujowh Kuram t Honologri b Lithy cr acaval
```

2nd-Order Markov Model – Conditioned on Letter Pairs
(i.e., estimated cond. probs. for a-z, A-Z & space)

```
>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

loft dick patace preend buff Cob Rit Goga co Kan brack hast kinc fieu
```

```
>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

te dump flip cam boup hoodeem tald twilt Hoyawn warchiz Boin Mact tid
```

```
>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

mock lubeate clavy dary ick sivarsart fown solavere dod gi Nage Luke
```

```
>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

wearn la sk Haass Ohin Chowl west se deub Guada molext heap dierk sai
```

```
>> char(gen_text_3(cond_probs_2,cond_probs_3,70,MAX))

walift Peny lue it cork faile gly wass Age jaxon Gaink hoy aiteck rit
```